1. Data Requirements and Functionalities

Our application comprises users as Caretakers or Pet Owners, as well as the PCS Administrators. For the Pet Owners, if the pet owner has not added any pet records to the website, the Pet Owners will not be able to use any functionality of the website such as showing availability and adding new bids. All pets and users must have an image uploaded when they register. Once their registration is complete, Pet Owners and Caretakers will be directed to their profile page where they can update their information accordingly. A Pet Owner can be a Caretaker at the same time if they register as a caretaker with the same username or vice versa. All Pet Owners and Caretakers must be at least 18 years old.

Pet Owners will be able to update information about their pets or add more records of their pets. Pet Owners will also be able to submit their bids and feedback for the bids that are successful. The bid they submitted must be under the availability period of the stated caretaker. After a bid has been set as successful, Pet Owners can access a form to settle transaction details including the mode of pet transfer, and the method of payment. The mode of transfer is restricted to three options: 1) pet owner deliver, 2) caretaker pick up, and 3) transfer through the physical building of PCS. Additionally, the method of payment is restricted to two options: 1) credit card, and 2) cash. When adding records of their pets, Pet Owners need to classify them under a category (which has already been specified in the system by the administrator). Pet Owners will be able to search for caretakers and other owners who are living nearby based on their postal code.

Once logged in, Caretakers will be able to see a calendar with their declared availability periods, and successful bids that they are involved in. Part-time caretakers will be able to input their available dates and they will be recorded in the database. Full-time caretaker's availability period will be automatically set to be every day unless they apply for leave. They must work a total of 2 X 150 consecutive days.

Both Pet Owners and Caretakers will be able to view their basic information such as their email and password and be able to update it. However, they will not be able to change their username. Caretakers may also view the number of pet-days they have taken up on the current month.

Administrators can login on a separate admin page where they can create an account as well. A user and an administrator cannot be logged in at the same time on the same browser. The person must log out from their user account to switch to the admin account and vice versa. After being logged into the administrators' page, they can add categories and set the base price for each category. All pets of all users need to be classified within these categories. They can also view the summary statistics such as the number of jobs, number of pets taken care of every month. The administrator will also be able to view the salary information for all caretakers.

2. Non-Trivial Implementation

2.1. Shorten Availability

Full-time caretakers can shorten their availability period by applying for leave. Part-time caretakers can shorten their declared availability period by updating or deleting it. When an availability period is shortened, the system will check that there are no successful bids that are no longer accounted for in the shortened period, before allowing this action. Otherwise, this action will return an error message.

2.2. Time zone

Each computer system has a different timezone, as set by the user. Each plugin only allows a limited range of time zones or Universal Time Zone (UTC). Hence, aligning the display of timestamps to the Singapore Time Zone (SGT) is a challenge. As a solution, all new timestamps will be converted to UTC and stored as such in the database. For plugins like Flatpickr that automatically displays the timestamps according to the time zone set by the user's computer, the timestamp will be manually changed to be displayed in UTC. For the application to work, the user's computer must be set to SGT time zone.

2.3. Profile Picture

A profile picture can be uploaded and stored in the database for the user and pets. The pictures can be changed at will, depending on the preference.

2.4. Rating Display

In addition, the display of ratings in the search caretaker function is converted into stars based on its percentage, allowing a more elegant display. This numeric rating value can be seen by all users.

2.5. Calculation of Caretakers' Salaries

The salaries of caretakers is calculated through a query, by retrieving relevant data from bids. They are also calculated differently for different types of users, such that full-time caretakers receive $3000 and a bonus according to their pet-days, and part-time caretakers receive 75% of all the total price of the bids they have taken in that month. If full-time caretakers have more than 60 pet days in the month, they will earn 80% of the whole salary they earnt in the month. If they have 60 or less petdays, they will earn the above mentioned $3000 for that month.

2.6. Enforcing Bids Constraints

Bids table references attributes from multiple entities so multiple triggers and checks have to be imposed to ensure that non-successful bids are differentiated from successful bids. For example, checks have to be enforced to ensure that rating/review- and transaction-related attributes are not updated for unsuccessful bids. Furthermore, since the bids table stores all bids that have been made so far, extra care must be taken to make sure that the statuses of bids that have been confirmed as successful or unsuccessful are not changed when updating the statuses of new bids. This is done by allowing the is_successful attribute to be null.

3. Data Constraints

3.1. Users

- Users are identified by their username
- A user must either be a caretaker or a pet owner (covering constraint)
- A user can be both a caretaker and a pet owner (overlapping constraint)
- Users must have a username and a password for identification and profile management purposes
- Password must be hashed and salted
- Users must have an email address to facilitate account recovery, notification of appointments and status updates of pets under caretaker's care
- Users must have a first and last name
- Users must have a date of birth to ensure that user is legally allowed to work as caretaker and/or request for services
- Users must have a postal code and optionally a unit number for pet pickup and searching nearby function
- Users must have a profile picture for identification verification purposes
- Users must register bank number for payment and salary transfer
- Users must have a registration date to track their accounts

3.2. Owners

- Owners are identified by their username
- Each owner must own at least one pet
- Owners without a pet would have its pet owner features disabled

3.3. Caretakers

- Caretakers are identified by their username
- Caretakers' rating out of 5 and the number of reviews they received must be recorded
- The rating and number of reviews must be updated every time a review is made
- Caretakers must either be full-time or part-time (covering constraint)
- Caretaker cannot be both full-time and part-time (non-overlapping constraint)
- Each caretaker must declare their availability
- Each caretaker gets a salary
- Each caretaker must charge a daily price for each pet
- Each caretaker can only take care of pets they can care for
- The number of pets that each caretaker can take care of (based on rating) must be recorded
- Each part-time caretaker cannot take care of more than 2 pets unless the rating is above 4 out of 5
- All caretakers must not take care of more than 5 pets at any one time

3.4. Availability

- Each availability period is identified by the caretaker username, start date and end date
- If each caretaker has availability periods with overlapping time slots, the periods will be automatically merged
- If a bid is successful, the availability period occupied by the successful bid cannot be changed (shrunk or deleted)
- Each part-time caretaker must be able to specify their availability for the current year and the next year
- Part-time caretakers cannot shorten or delete their availability period if there is at least one pet under their care, that is not accounted for when the period is shortened or deleted
- Each full-time caretaker is treated as available until they apply for leave
- Each full-time caretaker must work for a minimum 2 x 150 consecutive days a year
- Full-time caretakers cannot apply for leave if there is at least one pet under their care
- If a caretaker is deleted, his/her availability periods no longer need to be recorded in the database
- Each full-time caretaker is treated as available until they apply for leave, so when they register, an availability will be created automatically that ends one year form date of registration Thereafter, availability period of (1 year) for each caretaker must be added yearly

3.5. Pets

- Pets are identified by owner's username and pet name
- Pets must have a photo and size of identification purposes
- Pets may optionally have special requirements related to how they need to be taken care of (e.g. daily walk, types of food, etc.)
- Pets may optionally have a description and sociability level which can help caretakers provide better pet care services
- Every pet must be classified under exactly one category to facilitate browsing

3.6. Categories

- Categories are identified by their category name (cat_name)
- The daily price corresponding to each category must be recorded

3.7. Bids

- Each bid is identified by owner username, pet name, caretaker username, bid start timestamp, bid end timestamp, and availability start timestamp.
- Each bid also has availability end timestamp, rating, review, payment method, mode of transfer, a status indicating whether it is "successful" or "not successful", a status indicating whether it has been paid for or not, the total price of the bid, and the type of service of the bid
- Each owner can bid for a caretaker to take care of their pet, with a proposed bid start and bid end timestamp.
- The duration as indicated by the proposed start timestamp and proposed end timestamp of a bid must lie within an availability period of the caretaker being bid for
- The successful bid is chosen randomly by the system. is_successful attribute will be set to 'true' for this successful bid. If a bid is unsuccessful, the is_successful attribute is set to 'false'. If the system has not determined whether the bid should be chosen, the is_successful attribute will remain as NULL.
- Each bid, once successful, cannot be changed to its unsuccessful state
- Each owner may submit multiple reviews to the same caretaker if the caretaker has taken care of the owner's pet multiple times
- If a bid is unsuccessful, the rating, review, mode of transfer, method of payment, is_paid status must remain as NULL
- The bid end timestamp must be greater than the bid end timestamp

3.8. Timings

- Each timing is identified by start timestamp and end timestamp
- The end timestamp must be greater than the start timestamp

3.9. Charges

- Each charge is identified by caretaker username and category id
- For each charge, the daily price of the bid being charged must be recorded.

- Only successful bids will be charged
- Price for each category of each caretaker increases with the rating of the full-time caretaker but will never be below the base price
- Part-time caretakers must set their own daily price
- Full-time caretakers must use base price and rating to get daily price

3.10. Salaries

- Salaries are identified by caretaker's username, month, and year
- For each salary, the salary amount must be recorded.
- Part-time caretakers will receive 75% of their service price as salary.
- The salary of full-time Caretaker depends on how many Pet are taken care of in a given month for how many days (i.e. pet value day)
- Full-time Caretakers will receive a salary of $3000 per month for up to 60 pet-days.
- Full-time caretakers will receive 80% of the total money they earn in the month as bonus if they have more than 60 pet-days.
- If a caretaker is deleted, his/her salary no longer needs to be recorded in the database

3.11. Administrators

- Administrators have separate accounts from users since they are entitled to more privileges.
- Administrators are identified by their username.
- Administrators have a password.
- Passwords must be hashed and salted for security purposes
- Administrators set the pet category and the base daily price for each category, for full-time caretakers
- Salary is given by administrators to part-time and full-time caretaker monthly

4. ER Diagram

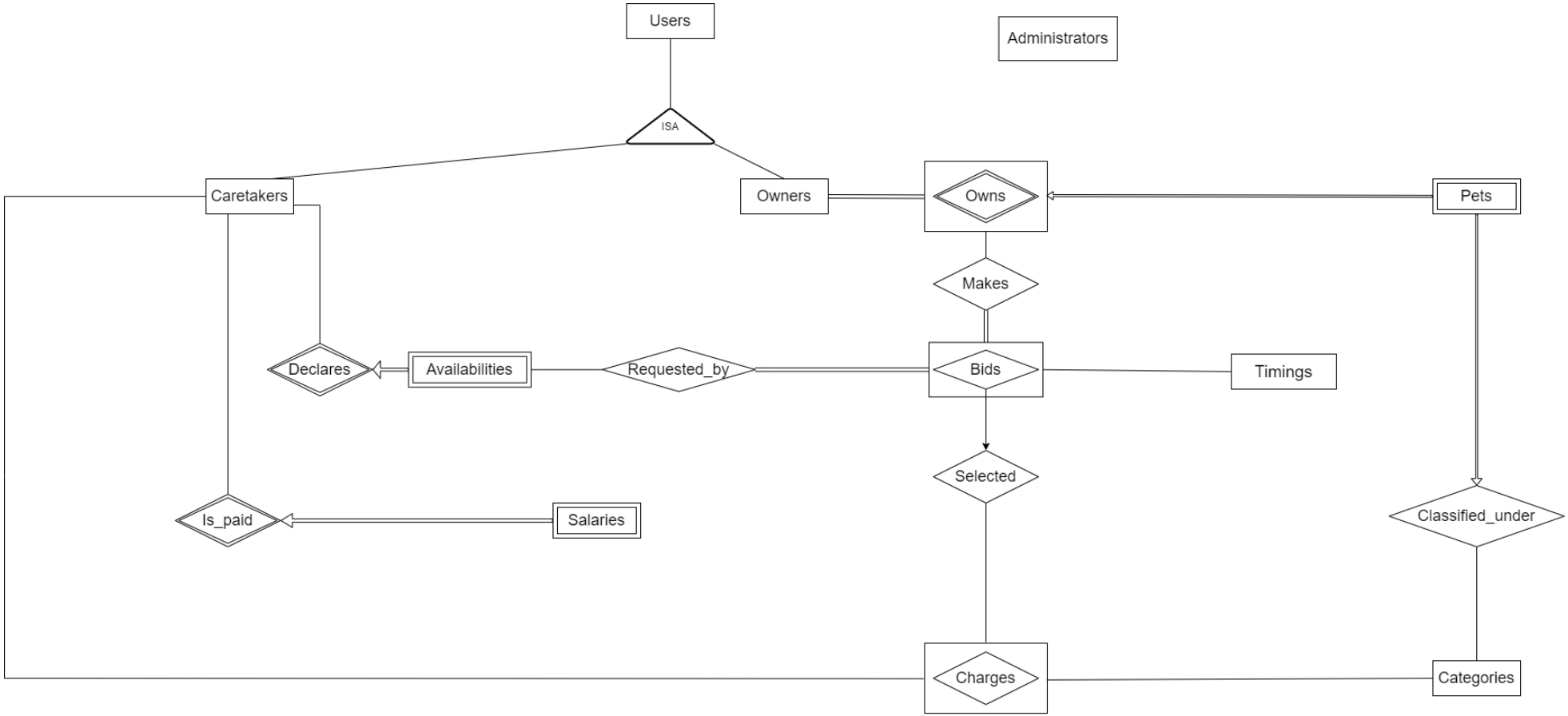


Figure 4-1: Refer to Appendix B for detailed ER diagram

4.1. Justifications for Non-Trivial Design Decisions

4.1.1. Derived Attributes

- Simplifies a lot of the calculation needed when retrieving information for the calculation of ratings.
- Derived attributes such as avg_rating and no_of_reviews, the avg_rating can be efficiently recalculated every time a new review is made. This is done by increasing the no_of_reviews by 1, adding the avg_rating with the new rating and getting the new avg_rating by dividing the calculated value with the new no_of_reviews
- total_price is also precalculated to facilitate the calculation of other values such as salaries and pet-value days without having to recalculating every time.
- Addition of Charges Table
- The Charges table consolidates the price charged by each caretaker for each category. We have added this since, full-time caretakers are free to specify a price other than the base-price. This is the case for part-time caretakers. Whether this daily price is more than or equal to the base price is checked by a trigger.

4.1.2. Addition of Timings Table

- The Timings table is created to allow the same pet owner to bid for the same availability period of the same caretaker for the same pet, but with different non-overlapping bided time periods.

4.2. Constraints Not Captured

4.2.1. Users

- A user must either be a caretaker or a pet owner (covering constraint)
- A user can be both a caretaker and a pet owner (overlapping constraint)
- Password must be hashed and salted

4.2.2. Owners

- Owners without a pet would have its pet owner features disabled

4.2.3. Caretakers

- Each part-time caretaker cannot take care of more than 2 pets unless the rating is above 4 out of 5
- All caretakers must not take care of more than 5 pets at any one time

4.2.4. Availability

- Each caretaker cannot have overlapping time slots
- Each part-time caretaker must be able to specify their availability for the current year and next year
- Each part-time caretaker cannot delete their availability if there is one pet under their care in this period
- Each part-time caretaker cannot update their availability if there is one pet under their care, that is unaccounted for in the new shortened availability period
- Each full-time caretaker must work for a minimum 2 x 150 consecutive days a year
- Each full-time caretaker is treated as available until they apply for leave
- Full-time caretakers cannot apply for leave if there is at least one pet under their care

4.2.5. Pets

- None

4.2.6. Categories

- None

4.2.7. Bids

- The time period as indicated by the proposed start date and proposed end date of a bid must lie within an availability period of the caretaker being bid for
- The ratings and review for a bid can only be recorded if the bid is successful
- The successful bid is chosen randomly by the system. is_successful attribute will be set to ‘true’ for this successful bid. If a bid is unsuccessful, the is_successful attribute is set to ‘false’. If the system has not determined whether the bid should be chosen, the is_successful attribute will remain as NULL.
- Each bid, once successful, cannot be changed to its unsuccessful state
- If a bid is unsuccessful, the rating, review, mode of transfer, method of payment, is_paid status must remain as NULL
- The bid end timestamp must be greater than the bid end timestamp

4.2.8. Charges

- Only successful bids will be charged
- Price for each category of each caretaker increases with the rating of the full-time caretaker but will never be below the base price
- Part-time caretakers must set their own daily price
- Full-time caretakers must use base price and rating to get daily price

4.2.9. Salaries

- Part-time caretakers will receive 75% of their service price as salary.
- The salary of full-time Caretaker depends on how many Pet are taken care of in a given month for how many days (i.e. pet-value day)
- Full-time Caretakers will receive a salary of $3000 per month for up to 60 pet-days.
- Full-time caretakers will receive 80% of their price as bonus for any excess pet-day.

4.2.10. Administrators

- Passwords must be hashed and salted for security purposes
- Administrators set the categories and the base daily price for each category, for full-time caretakers

5. Relational Schema

5.1. Users

```
CREATE TABLE Users (
username            VARCHAR      PRIMARY KEY,
first_name          NAME         NOT NULL,
last_name           NAME         NOT NULL,
password            VARCHAR(64)  NOT NULL,
email               VARCHAR      NOT NULL UNIQUE CHECK(email LIKE '%@%.%'),
dob                 DATE         NOT NULL CHECK (CURRENT_DATE - dob >= 6750),
credit_card_no      VARCHAR      NOT NULL,
unit_no             VARCHAR      CHECK (unit_no LIKE ('__-%') OR NULL),
postal_code         VARCHAR      NOT NULL,
avatar              BYTEA        NOT NULL,
reg_date            DATE         NOT NULL DEFAULT CURRENT_DATE,
is_owner            BOOLEAN      NOT NULL DEFAULT FALSE,
is_caretaker        BOOLEAN      NOT NULL DEFAULT FALSE
);
```

5.2. Owners

```
CREATE TABLE Owners (
username            VARCHAR      PRIMARY  KEY  REFERENCES  Users(username)  ON  DELETE
CASCADE,
is_disabled         BOOLEAN      NOT NULL DEFAULT TRUE
);
```

5.3. Caretakers

```
CREATE TABLE Caretakers (
username            VARCHAR      PRIMARY  KEY  REFERENCES  Users(username)  ON  DELETE
CASCADE,
is_full_time        BOOLEAN      NOT NULL,
avg_rating          FLOAT        NOT NULL DEFAULT 0,
no_of_reviews       INT          NOT NULL DEFAULT 0,
no_of_pets_taken    INT          CHECK (no_of_pets_taken >= 0) DEFAULT 0,
is_disabled         BOOLEAN      NOT NULL DEFAULT FALSE
);
```

5.4. Availabilities

```
CREATE TABLE declares_availabilities(
start_timestamp           TIMESTAMP          NOT NULL,
end_timestamp             TIMESTAMP          NOT NULL,
caretaker_username        VARCHAR            REFERENCES Caretakers(username) ON DELETE
CASCADE ON UPDATE CASCADE,
```

```
CHECK (end_timestamp > start_timestamp),
PRIMARY KEY(caretaker_username, start_timestamp) --Two availabilities belonging to the
same caretaker should not have the same start date
);
```

5.5. ownsPets

```
CREATE TABLE ownsPets (
username                    VARCHAR             NOT NULL REFERENCES Owners(username) ON
DELETE CASCADE,
name                        NAME                NOT NULL,
description                 TEXT,
cat_name                    VARCHAR(10)         NOT NULL REFERENCES Categories(cat_name),
size                        VARCHAR             NOT NULL CHECK (size IN ('Extra Small',
'Small', 'Medium', 'Large', 'Extra Large')),
sociability                 TEXT,
special_req                 TEXT,
img                         BYTEA,

PRIMARY KEY (username, name)
);
```

5.6. Categories

```
CREATE TABLE Categories (
cat_name                    VARCHAR(10)         PRIMARY KEY,
base_price                  NUMERIC
);
```

5.7. Bids

```
CREATE TABLE bids (
owner_username              VARCHAR,
pet_name                    VARCHAR,
bid_start_timestamp         TIMESTAMP,
bid_end_timestamp           TIMESTAMP,
avail_start_timestamp       TIMESTAMP,
avail_end_timestamp         TIMESTAMP,
caretaker_username          VARCHAR,
rating                      NUMERIC,
review                      VARCHAR,
is_successful               BOOLEAN,
payment_method              VARCHAR,
mode_of_transfer            VARCHAR,
is_paid                     BOOLEAN,
total_price                 NUMERIC             NOT NULL CHECK (total_price > 0),
type_of_service             VARCHAR             NOT NULL,

PRIMARY KEY (pet_name, owner_username, bid_start_timestamp, bid_end_timestamp,
caretaker_username, avail_start_timestamp),
FOREIGN KEY (bid_start_timestamp, bid_end_timestamp) REFERENCES
Timings(start_timestamp, end_timestamp),
FOREIGN KEY (avail_start_timestamp, caretaker_username) REFERENCES
declares_availabilities(start_timestamp, caretaker_username),
FOREIGN KEY (pet_name, owner_username) REFERENCES ownsPets(name, username),
UNIQUE (pet_name, owner_username, caretaker_username, bid_start_timestamp,
bid_end_timestamp);
CHECK ((is_successful = true) OR (rating IS NULL AND review IS NULL)),
CHECK ((is_successful = true) OR (payment_method IS NULL AND is_paid IS NULL AND
mode_of_transfer IS NULL))
CHECK ((rating IS NULL) OR (rating >= 0 AND rating <= 5))
```

```
CHECK ((bid_start_timestamp >= avail_start_timestamp) AND (bid_end_timestamp <=
avail_end_timestamp) AND (bid_end_timestamp > bid_start_timestamp))
);
```

5.8. Timings

```
CREATE TABLE Timings (
start_timestamp           TIMESTAMP,
end_timestamp             TIMESTAMP,

PRIMARY KEY (start_timestamp, end_timestamp),
CHECK (end_timestamp > start_timestamp)
);
```

5.9. Charges

```
CREATE TABLE Charges (
daily_price               NUMERIC,
cat_name                  VARCHAR(10)        REFERENCES Categories(cat_name),
caretaker_username        VARCHAR            REFERENCES Caretakers(username),

PRIMARY KEY (cat_name, caretaker_username)
);
```

5.10. Administrators

```
CREATE TABLE Administrators (
admin_username            VARCHAR            PRIMARY KEY,
password                  VARCHAR(64)        NOT NULL,
);
```

5.11. Constraints Enforced by Triggers

- check_pet_cover: a caretaker cannot take care of a pet that he cannot care for
- check_per_limit: any caretaker can only take care of a maximum of 5 pets at any point in time, and that any part-time caretaker that has an average rating of less than 4 can only take care of a maximum of 2 pets at any point in time
- check_daily_price: the daily price charged by any full-time caretaker for a particular category of pets cannot be less than the base price that has been set by the administrator for that category
- check_deletable_bid: bids that are either confirmed to be successful or have expired cannot be removed from the bids table
- check_deletable: an availability period can only be removed from the declares_availabilities table if there are no pets that the caretaker is scheduled to care for in that period
- merge_availabilities: If each caretaker has availability periods with overlapping time slots, the periods will be automatically merged
- update_availabilities: If a bid is successful, the availability period occupied by the successful bid cannot be changed (shrunk or deleted)
- update_disable: Owners without a pet would have its pet owner features disabled
- update_caretaker: A user must either be a caretaker, or a pet owner (covering constraint) and a user can be both a caretaker and a pet owner (overlapping constraint)
- update_owner: A user must either be a caretaker, or a pet owner (covering constraint) and a user can be both a caretaker and a pet owner (overlapping constraint)

5.12. Constraints Unenforced

- the price charged by a full-time caretaker increases with the rating

6. Normalisation

6.1. Categories

- Non-key attribute base_price is fully dependent on primary key cat_name
- Therefore, this table is in BCNF

6.2. Users

- Non-key attributes first_name, last_name, password, email, dob, credit_card_no, unit_no, postal_code, avatar, reg_date, is_owner and is_caretaker are fully dependent on primary key username
- Therefore, this table is in BCNF

6.3. Owners

- Non-key attribute is_disabled is fully dependent on primary key username
- Therefore, this table is in BCNF

6.4. Caretakers

- Non-key attributes is_full_time, avg_rating, no_of_reviews, no_of_pets_taken, is_disabled are fully dependent on primary key username
- Therefore, this table is in BCNF

6.5. ownsPets

- Non-key attributes description, size, sodiability, special_req, img are fully dependent on primary key (username, name)
- Therefore, this table is in BCNF

6.6. declares_availabilities

- Non-key attribute end_timestamp is fully dependent on primary key (caretaker_username, start_timestamp)
- Therefore, this table is in BCNF

6.7. Timings

- There are no non-key attributes
- Therefore, this table is in BCNF

6.8. Bids

- Non-key attributes avail_end_timestamp, rating, review, is_successful, payment_method, mode_of_transfer, is_paid, total_price, type_of_service are fully dependent on primary key (pet_name, owner_username, bid_start_timestamp, caretaker_username, avail_start_timestamp)
- Therefore, this table is in BCNF

6.9. Administrators

- Non-key attributes password is fully dependent on primary key admin_id
- Therefore, this table is in BCNF

6.10. Charges

- Non-key attribute daily_price is fully dependent on primary key (cat_name, caretaker_username)
- Therefore, this table is in BCNF

Note: Since it is required for a table to be 3NF before it is BCNF, all tables are in 3NF as well.

7. Triggers

7.1. Trigger 1

The function merge_availabilities merges 2 coinciding availability periods together. It is called before an availability is inserted into the table. The function will not be applied to full time caretakers, as they cannot manually add availabilities, only the system can do so. Hence, if there is a caretaker in the Caretakers table that has the same caretaker_username as the new availability period, then the function will return. Afterwards, it checks that the new availability period is valid, where start_timestamp is greater than end_timestamp. For the first case, it checks that the new availability does not coincide with the existing availability period, This is illustrated in Figure 7-1, if the greater of 2 period's start_timestamp is smaller than the lesser of 2 period end_timestamp, then 2 periods coincide. Hence if 2 periods meeting these conditions does not exist, no existing availability coincides with the new one. Therefore, the new period can be added without merging. For the second case, if the new period is a subset of an existing one, the new period does not have to be added. For the third case where 2 periods coincide, the existing period will be deleted, and the new merged period will be added.

```
Create Trigger
CREATE OR REPLACE FUNCTION merge_availabilities()
RETURNS TRIGGER AS
$$ DECLARE new_start TIMESTAMP WITHOUT TIME ZONE;
 new_end TIMESTAMP WITHOUT TIME ZONE;
 old_start TIMESTAMP WITHOUT TIME ZONE;
```

```
BEGIN
RAISE NOTICE 'Entering merge_availabilities function ...';

IF EXISTS (SELECT 1 FROM Caretakers WHERE NEW.caretaker_username = username AND
is_full_time IS TRUE) THEN
-- Do not need to merge availability for full-time caretakers as they do not insert
availability, but take leave instead
RETURN NEW;
END IF;

IF NEW.start_timestamp > NEW.end_timestamp THEN
RETURN NULL;
ELSIF
NOT EXISTS (SELECT 1 FROM declares_availabilities a1 WHERE a1.caretaker_username =
NEW.caretaker_username AND (GREATEST (a1.start_timestamp, NEW.start_timestamp) <= LEAST
(a1.end_timestamp, NEW.end_timestamp))) THEN --No overlap
RETURN NEW;
ELSIF
EXISTS (SELECT 1 FROM declares_availabilities a2 WHERE a2.caretaker_username =
NEW.caretaker_username AND (a2.start_timestamp <= NEW.start_timestamp AND
a2.end_timestamp >= NEW.end_timestamp)) THEN --New period is a subset of an existing
period
RETURN NULL;

ELSE --Two periods coincide
RAISE NOTICE 'Going to merge 2 periods ...';
SELECT LEAST(a3.start_timestamp, NEW.start_timestamp), GREATEST(a3.end_timestamp,
NEW.end_timestamp), a3.start_timestamp INTO new_start, new_end, old_start
FROM declares_availabilities a3
WHERE a3.caretaker_username = NEW.caretaker_username AND (GREATEST (a3.start_timestamp,
NEW.start_timestamp) <= LEAST (a3.end_timestamp, NEW.end_timestamp));

DELETE FROM declares_availabilities a4
WHERE a4.caretaker_username = NEW.caretaker_username AND a4.start_timestamp = old_start;
RETURN (new_start, new_end, NEW.caretaker_username);
END IF;
END $$
LANGUAGE plpgsql;
```

Call Trigger

```
CREATE TRIGGER merge_availabilities
BEFORE INSERT ON declares_availabilities
FOR EACH ROW EXECUTE PROCEDURE merge_availabilities();
```

7.2. Trigger 2

Update_availabilities checks if the availability can be updated and updates it according to user input.

When the availability is shortened (as compared to original), it checks whether there are successful bids that are not accounted for, in the shortened availability. As shown in Figure 7-1, when 2 periods coincide, the greater of the start_timestamp in these 2 periods will be smaller than the lesser of end_timestamp in these 2 periods. Therefore, the system checks whether any bids coincide with the period that is no longer existent when availability is shortened.

As seen in figure 7-2, this check is first done on the left part and indicates first_result := 1 if there are successful bids coinciding with the left part. It is then done on the right part and indicates second_result := 1 if there are successful bids coinciding with the right part.

The first_result and second_result will add up to be the total_result. If the total_result is 0, means there are no successful bids between the left and right part that are no longer accounted for. The bid will then proceed to update. In the other cases, an exception will be raised, preventing the bid form updating.

Create Trigger

```
CREATE OR REPLACE FUNCTION update_availabilities()
RETURNS TRIGGER AS
$$ DECLARE first_result INTEGER := 0;
  DECLARE second_result INTEGER := 0;
```

```
  DECLARE total_result INTEGER;
  BEGIN
  IF (OLD.start_timestamp < NEW.start_timestamp) THEN
  IF EXISTS(SELECT 1
            FROM bids b1
             WHERE b1.caretaker_username = OLD.caretaker_username
  AND  GREATEST(b1.bid_start_timestamp, OLD.start_timestamp)
  < LEAST(b1.bid_end_timestamp, NEW.start_timestamp))
  AND b1.is_successful IS TRUE
  THEN
  RAISE NOTICE 'there is a bid between the old and new starting timestamp';
       first_result := 1;
       ELSE
       first_result := 0;
       END IF;
        END IF;

IF (OLD.end_timestamp > NEW.end_timestamp) THEN
IF EXISTS(SELECT 1
          FROM bids b2
          WHERE b2.caretaker_username = OLD.caretaker_username
          AND GREATEST(b2.bid_start_timestamp, NEW.end_timestamp)
          < LEAST(b2.bid_end_timestamp, OLD.end_timestamp))
          AND b1.is_successful IS TRUE
       THEN
       RAISE NOTICE 'there is a bid between the old and new ending timestamp';
       second_result := 1;
       ELSE
       second_result := 0;
       END IF;
       END IF;

   total_result := first_result + second_result;
   IF total_result = 0 THEN
   RETURN NEW;
   ELSE
   RAISE EXCEPTION 'Availability cannot be updated as there as pet caring
   tasks in the original availability period';
   END IF;
   END $$
  LANGUAGE plpgsql;
```

Call Trigger

```
CREATE TRIGGER update_availabilities
BEFORE UPDATE ON declares_availabilities
FOR EACH ROW EXECUTE PROCEDURE update_availabilities();
```

7.3. Trigger 3

This trigger checks for the total number of successful bids for a particular caretaker that either have not started, or not ended yet, and assigns it to the variable ctx. It then computes the caretaker's average rating and assigns it to the variable rate. Lastly, it checks whether the caretaker is part-time by querying the number of rows that contain the caretaker's username as well as a false value for the is_full_time attribute, and assigning this number to the variable is_part_time. If the variable ctx has a value that is more than or equal to 5, it indicates that the pet limit has been reached, thus the caretaker is currently not able to care for any more pets and the row is not updated or added. Otherwise, if the caretaker is part-time (indicated by the variable is_part_time having a value that is greater than 0), his rating is either null (which indicates that no rating has been given for that caretaker so far) or less than 4, and he is currently caring for at least 2 pets, then the row will not be updated or added.

Create Trigger

```
CREATE OR REPLACE FUNCTION is_pet_limit_reached() RETURNS TRIGGER AS
$$ DECLARE ctx NUMERIC;
DECLARE rate NUMERIC;
DECLARE is_part_time NUMERIC;
BEGIN
```

```
SELECT COUNT(*) INTO ctx FROM bids B WHERE B.is_successful = true AND
B.caretaker_username = NEW.caretaker_username AND B.bid_end_timestamp >=
CURRENT_TIMESTAMP;
SELECT AVG(rating) INTO rate FROM bids Bo WHERE Bo.is_successful = true AND
Bo.caretaker_username = NEW.caretaker_username;
SELECT COUNT(*) INTO is_part_time FROM Caretakers C WHERE C.username =
NEW.caretaker_username AND is_full_time = false;
IF (ctx >= 5) THEN RETURN NULL; END IF;
IF (is_part_time > 0 AND ctx >= 2 AND (rate IS NULL OR rate < 4)) THEN RETURN NULL; END
IF;
RETURN NEW;
END; $$
LANGUAGE plpgsql;
```

Call Trigger

```
CREATE TRIGGER check_pet_limit
BEFORE INSERT OR UPDATE ON bids
FOR EACH ROW EXECUTE PROCEDURE is_pet_limit_reached()
```

8. Queries

8.1. Query 1

The query below is take_leave which is executed when a full-time caretaker applies for leave with the 'Apply for Leave' button of the 'My Availabilities' page. take_leave takes in a start and end timestamp for the leave period, as well as the caretaker's username.

Firstly, it checks that there are successful bids under the leave period. This is done by checking that the bid_start_timestamp is within the leave period, for each bid. If there are no successful bid in the period proceed.

Secondly, it obtains the start timestamp for the leftover availability period on the left, avail_first_start. The same is done to obtain the start and end timestamp for the leftover period on the right, avail_second_start and avail_second_end.

Thirdly, it checks that the caretaker works for at least 2 X 150 consecutive days. This is done by finding the interval in days, of the 2 leftover availability periods. The same calculation is done for all other availability period under the caretaker. Each interval will be converted to period type as follows:

- Period type 0: Interval < 150 days
- Period type 1: Interval >= 150 days and interval < 300 days
- Period type 2: interval >= 300 days

All these period type will be added into the result variable. If result < 2, the condition of 2 X 150 days is not fulfilled, so an exception is raised.

Otherwise, all the bids that are referencing the left (or first) leftover availability period will have their end timestamp updated to leave start timestamp (leave_start_timestamp). The left availability period will similarly. have their end timestamp updated

On the other side, a new updated right availability period will be inserted. All the bids referencing the right availability period will have their referencing (avail_start_timestamp) updated. Then the old availability period is deleted. This is illustrated in Figure 8-1.

Query

```
take_leave: 'CALL take_leave($1::timestamp AT TIME ZONE \'UTC\', $2::timestamp AT TIME
ZONE \'UTC\', $3)',
```

Stored Procedure

```
CREATE OR REPLACE PROCEDURE take_leave(leave_start TIMESTAMP WITH TIME ZONE, leave_end
TIMESTAMP WITH TIME ZONE, username VARCHAR) AS
$$ DECLARE avail_first_start TIMESTAMP;
DECLARE avail_second_start TIMESTAMP;
DECLARE avail_second_end TIMESTAMP;
DECLARE consecutive_days_first NUMERIC;
DECLARE consecutive_days_second NUMERIC;
DECLARE result NUMERIC;

BEGIN
```

```
result := 0;
RAISE NOTICE 'in take_leave procedure';
IF EXISTS (SELECT 1
             FROM bids
             WHERE caretaker_username = username
             AND bid_start_timestamp >= leave_start
                    AND bid_start_timestamp <= leave_end
                    AND is_successful IS TRUE) THEN
RAISE EXCEPTION 'Leave cannot be taken as there are scheduled pet-care jobs within the
leave period';
END IF;

SELECT start_timestamp INTO avail_first_start
FROM declares_availabilities
WHERE caretaker_username = username AND leave_start > start_timestamp AND leave_start <
end_timestamp;

SELECT start_timestamp, end_timestamp INTO avail_second_start, avail_second_end
FROM declares_availabilities
WHERE caretaker_username = username AND leave_end < end_timestamp AND leave_end >
start_timestamp;

SELECT DATE_PART('day', leave_start - avail_first_start) INTO consecutive_days_first;
SELECT DATE_PART('day', avail_second_end - leave_end) INTO consecutive_days_second;

WITH days_interval AS
(SELECT DATE_PART('day', end_timestamp - start_timestamp) AS days
FROM declares_availabilities
WHERE caretaker_username = username
       AND start_timestamp < avail_first_start
       OR start_timestamp > avail_second_start)

SELECT COALESCE(SUM(CASE
             WHEN days < 150 THEN 0
             WHEN days >= 150 AND days < 300 THEN 1
             WHEN days >= 300 THEN 2
             END), 0) INTO result FROM days_interval;
 RAISE NOTICE '1. result is %', result;

IF (consecutive_days_first >= 300) THEN
 result := result + 2;
ELSIF (consecutive_days_first >= 150)  THEN
 result := result + 1;
 END IF;
 RAISE NOTICE '2. result is %', result;

IF (consecutive_days_second >= 300) THEN
 result := result + 2;
 ELSIF (consecutive_days_second >= 150)  THEN
 result := result + 1;
 END IF;

  RAISE NOTICE '3. result is %', result;

IF (result < 2) THEN
       RAISE EXCEPTION 'Cannot take leave as you are not working for 2 X 150 consecutive
days this year';
END IF;

UPDATE bids SET avail_end_timestamp = leave_start WHERE bid_start_timestamp >=
avail_first_start AND bid_end_timestamp <= leave_start;
UPDATE declares_availabilities SET end_timestamp = leave_start WHERE start_timestamp =
avail_first_start;
INSERT INTO declares_availabilities VALUES (leave_end, avail_second_end, username);
```

```
UPDATE bids SET avail_start_timestamp = leave_end WHERE bid_start_timestamp >=
avail_second_start AND bid_end_timestamp <= avail_second_end;

IF (avail_first_start <> avail_second_start) THEN
DELETE FROM declares_availabilities WHERE start_timestamp = avail_second_start;
END IF;
END; $$
LANGUAGE plpgsql;
```

8.2. Query 2

This query calculates the salary of each caretaker. The query does a natural join on Bids and caretakers to have access to is_full_time attribute. Based on which, it does the calculation of salaries in 3 cases. First, for caretakers that are full time and have pet days more than 60. Second for caretakers that are full time and have pet days 60 and less. Finally, for caretakers that are part time.

These salaries are then grouped by their year and month and the full-time attributes.

```
SELECT
  extract(year from B2.bid_start_timestamp) as year,
  extract (month from B2.bid_start_timestamp) as month,
  B2.caretaker_username,
  SUM(b2.bid_end_timestamp - b2.bid_start_timestamp) as pet_days,
  CASE
    WHEN c.is_full_time AND
        SUM(b2.bid_end_timestamp - b2.bid_start_timestamp) > 60
             THEN 3000 + SUM (total_price)*0.8
    WHEN c.is_fulltime AND
        SUM(b2.bid_end_timestamp - b2.bid_start_timestamp) <= 60
             THEN 3000
    ELSE SUM (total_price)* 0.75
  END AS salary
  -- Inner join to get is_full_time attribute
FROM bids B2 INNER JOIN caretakers C on C.username = B2.caretaker_username
WHERE B2.is_successful AND pet_days <= 60
GROUP BY year, month, c.is_full_time, caretaker_username
ORDER BY year DESC, month DESC, caretaker_username ASC;
```

8.3. Query 3

This procedure takes in as parameters the owner's username (ou), pet's name (pn), timestamp for the start of the proposed service period (ps), timestamp for the end of the proposed service period (pe), timestamp for the start of a caretaker's availability period (sd), timestamp for the end of the same availability period (se), the caretaker's username (ct), and the type of service (ts). To calculate the total price of the service, the number of days in the proposed service period is multiplied by the price that the caretaker charges for the category under which the pet is under. The category that the pet is under is queried from the ownsPet table by searching for the row which contains the owner's username and the pet's name. Since it is cumbersome for the user to manually input the timestamps for the start and end of a proposed service period into the Timings table, given that the same information is also contained in the bids table, this procedure inserts this information into the Timings table whenever a bid corresponding to the proposed service period is inserted into the bids table. Lastly, once the bid has been inserted, its status, that is, whether it is successful or unsuccessful, will be updated on a random basis. Since only bids whose status is NULL will be updated, older bids whose status has been updated as successful or unsuccessful will no longer be updated in future.

Query

```
insert_bid: 'CALL insert_bid($1, $2, $3, $4, $5, $6, $7, $8)'
```

Stored Procedure

```
CREATE OR REPLACE PROCEDURE insert_bid(ou VARCHAR, pn VARCHAR, ps TIMESTAMP, pe
TIMESTAMP, sd TIMESTAMP, ed TIMESTAMP, ct VARCHAR, ts VARCHAR) AS
$$ DECLARE tot_p NUMERIC;
BEGIN
SELECT DATE_PART('day', pe - ps) INTO tot_p;
tot_p := tot_p * (SELECT daily_price FROM Charges WHERE caretaker_username = ct AND
cat_name = (SELECT cat_name FROM ownsPets WHERE ou = username AND pn = name));
```

```
IF NOT EXISTS (SELECT 1 FROM TIMINGS WHERE start_timestamp = ps AND end_timestamp = pe)
THEN INSERT INTO TIMINGS VALUES (ps, pe); END IF;
INSERT INTO bids VALUES (ou, pn, ps, pe, sd, ed, ct, NULL, NULL, NULL, NULL, NULL, NULL,
tot_p, ts);
UPDATE bids SET is_successful = (CASE WHEN random() < 0.5 THEN true ELSE false END)
WHERE is_successful IS NULL;
END; $$
LANGUAGE plpgsql;
```

9. Software Tools and Frameworks

In this project, we used ExpressJS of NodeJS with the Model-View-Controller (MVC) framework. The backend database is run on PostgreSQL and the application is hosted on Heroku. Our main form of communication and the meetings held are on Discord and Telegram.

10. Screenshots

Figure 10.1 shows the page of pets for the owner role. The sidebar also shows the panels available to the pet owner role. The navbar shows the search bar and the logout dropdown under the picture. Meanwhile, Figure 10.2 shows the administrator panel when signed in as administrator. The Admin settings page shows the different tabs each with different statistics. Finally, Figure 10.3 shows the calendar page that shows bids that are successfully accepted.

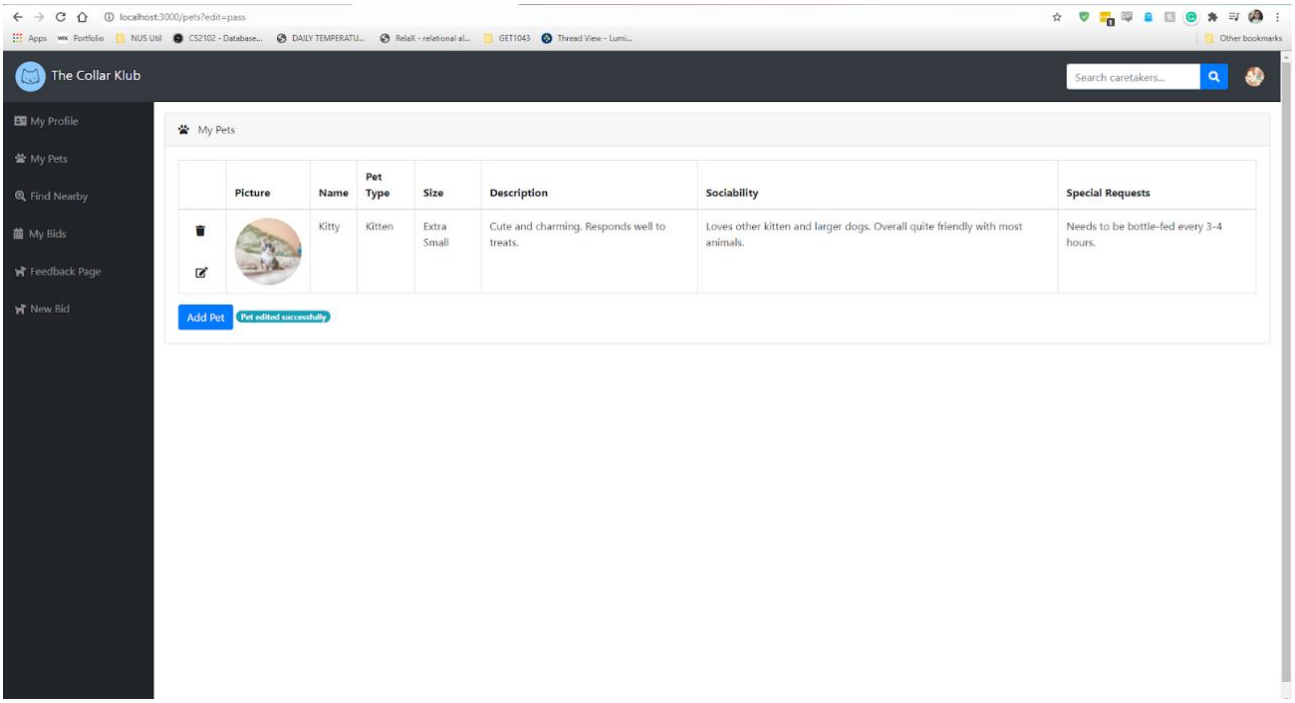


Figure 10-1: Pet profile page

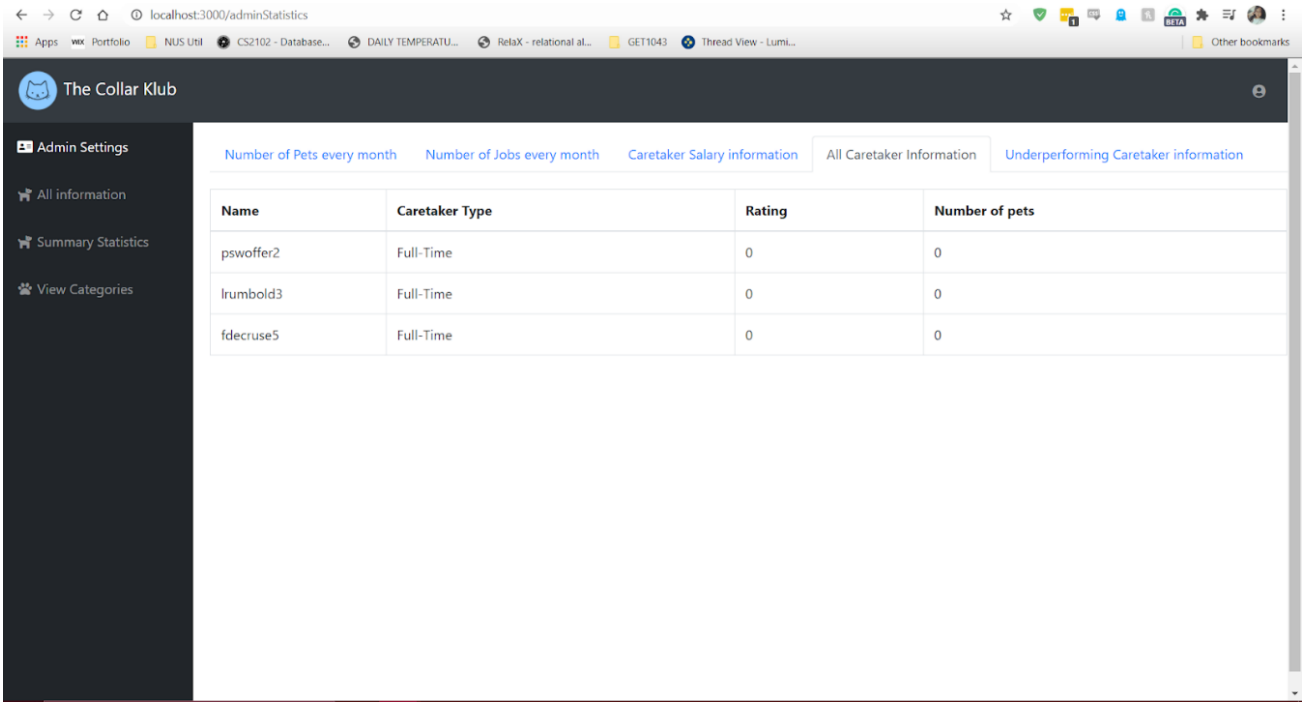


Figure 10-2: Administrator profile page

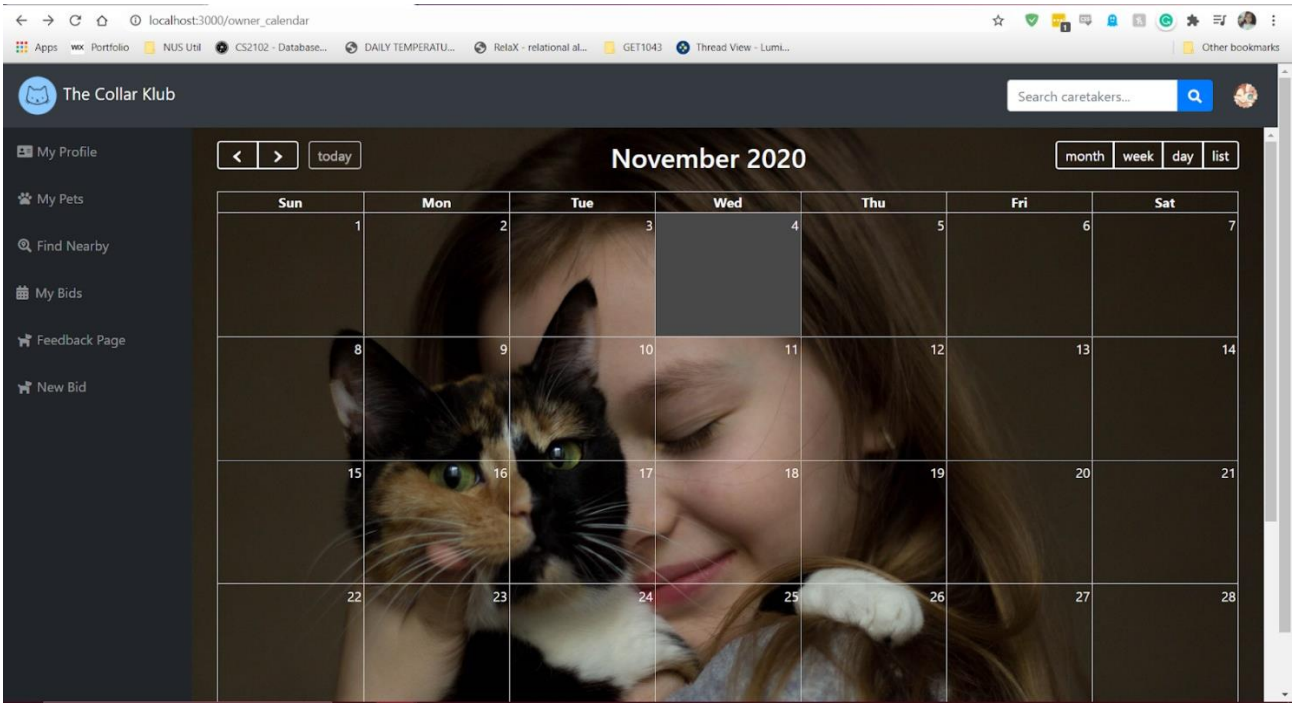


Figure 10-3: View availabilities/bids page

11. Learning Points

11.1. Lack of Experience

All of us did not have previous experience with ExpressJS framework. Hence getting started with the programming aspect and learning how to utilise this framework was interesting and challenging at the start. For instance, some of us who are foreign to the framework had difficulty understanding the base structure and getting started on the code. How were not sure of how to integrate the view pages with the controller and the subsequent database.

In order to resolve this issue, our team members bounced off each other to share tips and knowledge on how to use ExpressJS which helped the whole team to get familiar with the framework quickly and concentrate on implementing the various features. We share tips on some of the errors that we encountered prior and explain on what each page and code does. Nikhila especially guided us on how to properly set up a git and get to ensure that we were merging, pushing and pulling when needed. And the rest of us help to debug different errors when we have different implementation or database related errors.

11.2. New Language

For some of us, Javascript is a completely new development language. We had to learn not just the framework ExpressJS (NodeJS), but the base language in itself, which took more than a week to achieve. The front-end languages were also foreign to some of us, taking a while to understand the differences between Jade, HTML and EJS. Ultimately, we decided to use HTML as a base language since it is the most familiar to most of us. The integration of expressJS to HTML was also challenging to some of us as we were unfamiliar with the syntax (<%= %>) and the escape characters.

To overcome this issue, we scour through the Internet and search engines to understand and find templates that demonstrate how all these codes work. Eventually, it became rather straightforward for most of us, who can then teach the rest on how to display certain information.

11.3. Communication

In light of the recent COVID-19 situation, we were unable to physically meet up for our meetings. Some of us prefer the traditional style of sitting down on a round table to discuss the code and project specifications, which we think are more efficient in tracking progress and allow easier communication. Therefore, with this new restriction, communication was a little more difficult and each person's progress was more difficult to track.

To resolve this issue, Xin Ru set up a discord server for meetings, tracking of information and general one-to-one communication. We meet every week to discuss progress and check the current status of the application, as well as checkpoint the progress for the next meeting. To have a centralised location for tracking idea development and retaining information mimics the face-to-face meetings as much as possible. We held private one-to-one to better integrate the two separate features each of us were assigned to do. In the end, the communication was still effective enough to keep everyone on the same page.

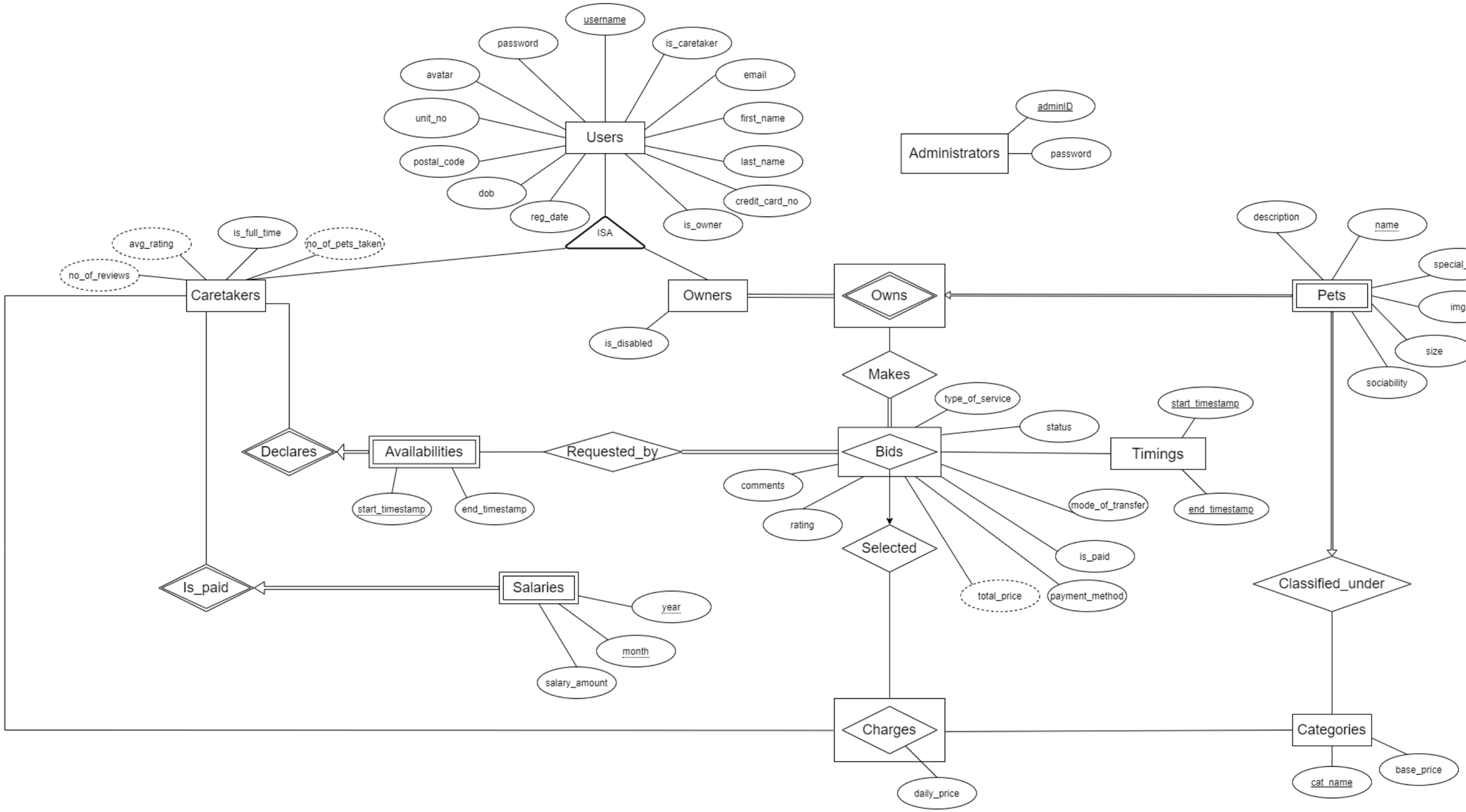
11.4. Miscellaneous

One thing our team learnt was to determine if there is a real necessity for an entity in the ER Diagram. Initially our move was to split a particular entity into many different entities as it made sense in a real-life scenario. However, when we received comments, we realised that it could all be merged into one entity. This was eye-opening for us as we learnt to see how to represent entities as in real world scenarios while making sure there is a real necessity for the entity.

12. Appendix A – Team Members Contribution

Member Name	Responsibilities
Phianne Calista Io	○ Created schema for bids, timings and charges tables ○ Created triggers check_daily_price, check_pet_cover, check_pet_limit, check_enddate ○ Created forms to insert new bids, give rating/review for caretakers, and settle transaction details for successful bids ○ Created queries related to bids
Leow Xiao Xuan	○ Created registration page for caretakers, and wrote related queries ○ Created a page and wrote queries for caretakers to view their summary information ○ Created page for caretakers to view their reviews
Yap Jia Aun	○ Set up schema for declare_availabilities table ○ Wrote queries for creating, reading, updating and deleting availability periods for part-time caretakers. ○ Wrote queries for automatic creation of 2 yearlong availability period for full-time caretakers. ○ Wrote queries for reading of availability periods and taking leave for full-time caretaker. ○ Crafted triggers for merging availabilities, checking that availability is deletable ○ Crafted triggers for checking that availability can be updated and that leave can be applied ○ Used Flatpickr timestamp selection plugin the front-end selection of availability period's start and end timestamp. ○ Used Full Calendar plugin to display availability and bids for owners and caretakers. ○ Created the frontend display for reading of bids and deletion of bids, on the owner's side.
Seh Xin Ru	○ Did the database of Owners, Pets and Categories (including its design, procedures, triggers, etc.) ○ Did the adding and editing of category in the admin panel ○ Set up the registration and authentication for users ○ Implemented the creation, display, updating and deletion of owners and pets ○ Implemented the addition, displaying and updating and anything else related to pictures for both Users and Pets ○ Implemented the location (proximity) search for both users and caretakers ○ Implemented the search of caretakers based on filtering and string search (including star display for ratings) ○ Implemented the dashboard display based on caretaker or owner or both roles ○ Designed the index page
Udayagiri Nikhila Sai	○ Set up separate authentication for users and administrators ○ Implement features involving creation, update and deletion of admin account in both frontend and backend ○ Set up frontend page to view all summary information of number of jobs, number of pets taken care in each month ○ Wrote queries to retrieve summary information for administrators which includes the pets in the month as well as the number of jobs in the month and the salary calculation ○ Calculate the salaries for all caretakers
Whole Team	○ We crafted the ER diagram according to application requirements. ○ We listed the constraints illustrated by the ER diagram, and preliminary constraints that were not shown. ○ We wrote up the project report and demonstration video.

13. Appendix B – Detailed ER Diagram



14. Appendix C – Diagrams

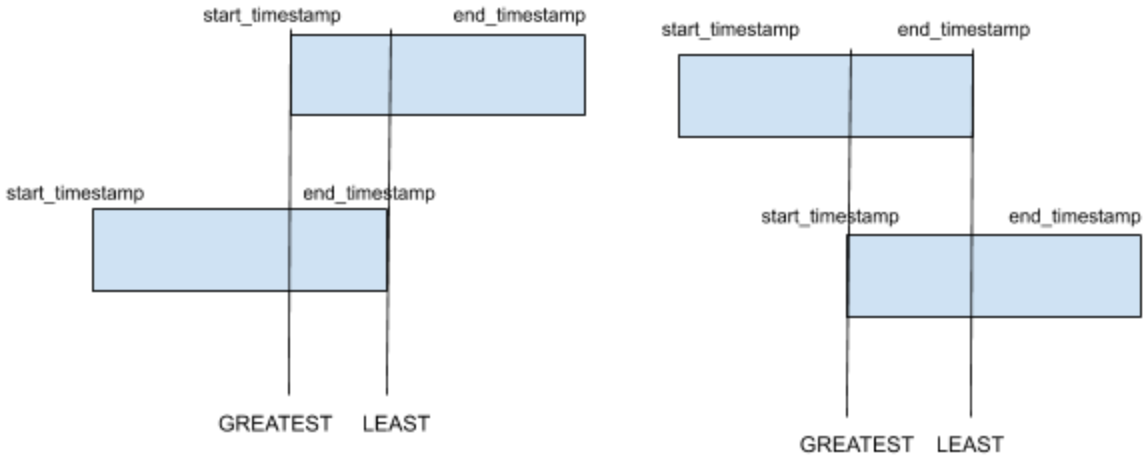


Figure 14-1: Finding if 2 periods collide

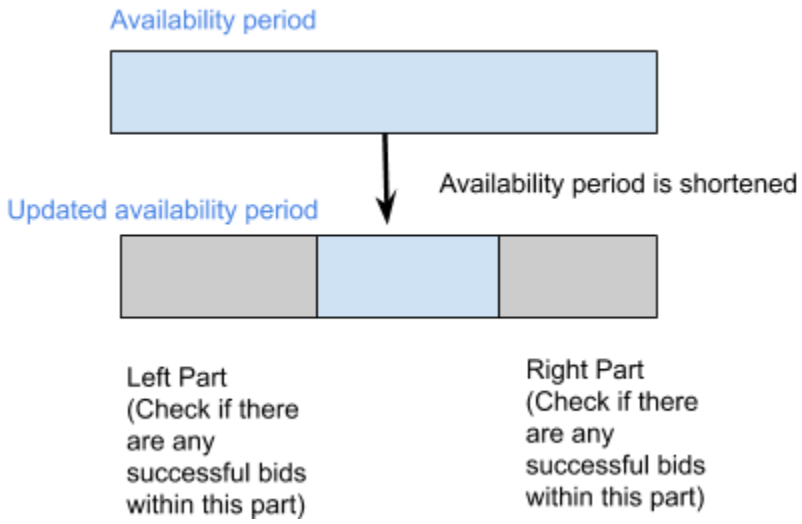


Figure 14-2: Left and right part of shortened availability period

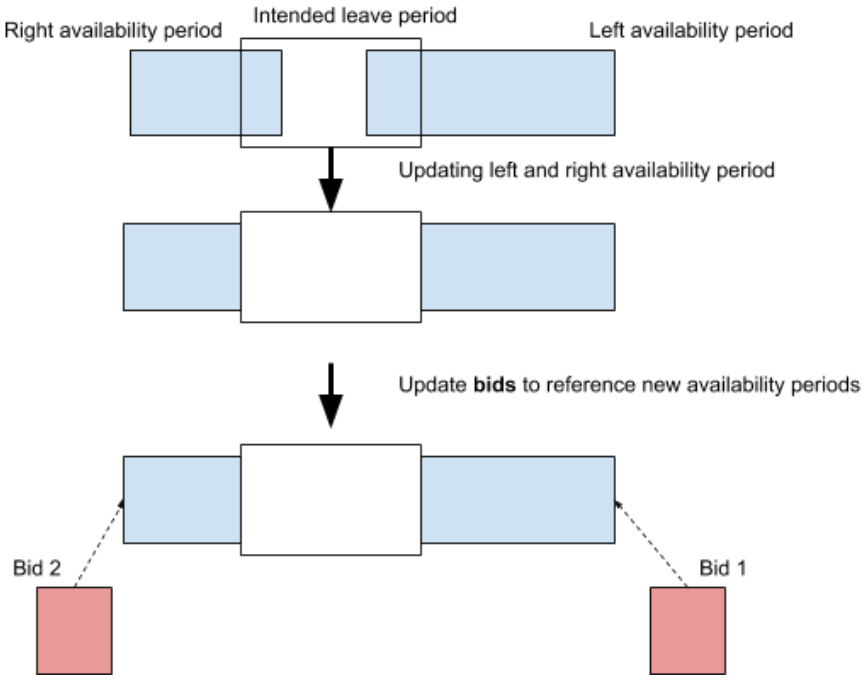


Figure 14-1: Updating availability periods